DII.3200.HP1020.P1.PM-1

Defense Information Infrastructure (DII)

Common Operating Environment (COE)

Programmer's Manual
for the Kernel Version 3.2.0.0 Patch 1
and Developer's Toolkit Version 3.2.0.1

(HP-UX 10.20)

September 26, 1997

Prepared for:

Defense Information Systems Agency

Prepared by:

Inter-National Research Institute (INRI)
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 20191

Table of Contents

List of Tables

List of Figures

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
`Courier Font`	Used to indicate entries to be typed at the keyboard, operating system commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: `tar xvf /dev/rmt/3mn`
Italics	Used for emphasis.

This page intentionally left blank.

1. Writing Programs Using the COE Tools

1.1 Overview

This document provides an introduction to the capabilities of the Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.2.0.0 tools for the HP-UX 10.20 Operating System. These tools consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. This document explains the basic use of the tools, regardless of whether they are run from a menu or from the command line. The document consists of the following sections and appendices:

1.2 Referenced Documents

The following documents are referenced in this programmer's manual:

- DII COE I&RTS:Rev 3.0, *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification Version 3.0*, July 1997

- DII.3200.HP1020.RG-1, *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.2.0.0 Application Programmer Interface (API) Reference Guide (HP-UX 10.20)*, July 25, 1997.

2. Application Development Overview

Developers may require access to public APIs to ensure an application complies with the *DII COE Integration and Runtime Specification*. To use the public APIs, developers must compile and link the application with the libraries and header files provided in the Developer's Toolkit.

NOTE: Remember that your `DII_DEV` directory and Motif `include` files and libraries may reside in a different location than in the example compile statements.

Public APIs are documented in the *DII COE API Reference Guide (HP-UX 10.20).*

2.1 Writing Your Application with the DII COE APIs

To access the DII COE tools through the provided APIs, you must include the following header in your application:

```
#include <DIITools.h>
```

The standard location for the Developer's Toolkit header is:

```
DII_DEV/include
```

The following is an example of using the COEAskUser tool, which is used to display a question and two possible responses to the user. After the user chooses a response, the response is returned.

```
#include <stdio.h>
#include <DIITools.h>

/*****************************/
/* COEAskUser_example        */
/*****************************/
int main(int argc, char *argv[])
{
char     b1_lab[] = "MY_YES";
char     b2_lab[] = "MY_NO";
char     message[]="This is my test Message";
int      ret_val;

/* Call DII/COE Library Function */
ret_val = COEAskUser, message, b1_lab, b2_lab);

exit(ret_val);
}
```

2.2 Building Your Application with the DII COE APIs

To build your application with DII COE APIs, you must link your application with the `libCOETools.a`, `libCOE.a`, and `libPrintClient` libraries, which are on the DII COE Developer's Toolkit tape.

The standard location for the Developer's Toolkit libraries is:

```
DII_DEV/libs
```

The actual compile and link statement for an application that uses DII COE APIs should resemble the following (substitute your location for the `DII_DEV` directory and Motif libraries and `include` files):

```
cc -Aa -o COEAskUser_example COEAskUser_example.c -I/h/DII_DEV/include
-I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools
-lCOE -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11
```

where `COEAskUser_example` is the name of the program being compiled.

2.3 Running Your Application

The DII COE provides the foundation and infrastructure in which one or more applications run. To operate under the COE, applications must be formatted properly as segments. The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are managed most conveniently as a unit. Segments generally are defined to keep related CSCIs together so functionality easily may be included or excluded. All applications must be put in the DII COE runtime environment segment format to be installed onto a DII COE-compliant machine. Refer to Section 4, *Segment Development*, for more information about segment development.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, `SegDescrip`, which is called the segment descriptor subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

Once installed, your application can be invoked in the DII COE environment in two ways: (1) running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 4.4, *Customizing Your Segment*, describes how to set up your application to be invoked as a menu item or as an icon.

3. Printer Overview

3.1 COE Print Service

The DII COE Printer API provides a simple, platform-independent method for DII COE applications to print text and graphics data. The API currently consists of 12 C language functions and 3 executable programs.

The COE print service is based on a client-server architecture. Each printer is managed by a single workstation that acts as the server for all print requests for that printer. The print server handles access controls, queue management, and error notification.

Every COE workstation runs a "printer agent," which facilitates communication between client applications and the print server. All printer API functions and executable programs use this printer agent.

3.2 Accessing Printers Using DII COE API

The DII COE Printer API consists of the following 12 C language low-level and high-level functions.

NOTE: Low-level and high-level functions should not be used within the same application.

Low-level Functions

```
C  int close_printer(char **file_name, FILE **file_pointer);

C  int get_printer_descriptions(char **c_printer_description);

C  int get_printer_name(char **c_printer_name);

C  int get_printer_type(char **c_printer_type);

C  int open_printer(char *xcp_security_level,
   int xi_line_length,
   int xi_page_length,
   int xi_line_spacing,
   int xi_indent,
   char **xcp_file_name,
   FILE **xfp);

C  int page_break(FILE **xfp);

C  int write_printer(char **c_string, FILE *fp);

C  int write_printer_array(char **c_string, FILE *fp);
```

High-level Functions

```
C  VDirectPrintFile(char *filename, int prt_rec)
C  VDirectPrintMsg(**msg_array, int nlines)
C  VPrintFile(char *filename)
C  VPrintMsg(char **msg_array, int nlines)
```

The `close_printer` function is used to conclude a print job and send the data to the printer. If `close_printer` is not called, the print job will not print.

The `get_printer_descriptions`, `get_printer_name`, and `get_printer_type` functions allow an application to retrieve the name, the type, and a description of the current default printer. All three functions return a string value via the pointer that was passed as an argument to the function. The printer name and description are simple text fields. The printer type is "ASCII", "HPCL", or "PostScript".

The `open_printer` function is used to send text data to a printer. It establishes a print context, including the security level, line length, page length, line spacing, and indentation for the print job. It returns a file pointer through its last argument. This file pointer is used for all subsequent actions on this print job.

The `page_break` function is used to indicate that the lines of text that follow should begin at the top of the next page.

The `write_printer` and `write_printer_array` functions are used to send the actual text data to a previously opened printer context.

The `VDirectPrintFile` and `VPrintFile` functions are used to print text data from a file on disk. The `VDirectPrintMsg` and `VPrintMsg` functions are used to print text data from an array of strings in memory. All four functions generate a security banner at the top and bottom of each output page. On completion, these functions return the internal number of the selected printer or they return `-1` if the user canceled the job or if an error occurred.

The `VPrintFile` and `VPrintMsg` functions provide the user with a `Print Chooser` window, which allows the user to select the destination printer. The `VDirectPrintMsg` function bypasses the `Print Chooser` window and prints directly to the COE default printer. The `VDirectPrintFile` function bypasses the `Print Chooser` window and prints directly to the specified printer. The printer number is specified as a return value from a previous `VPrintXXX` or `VDirectPrintXXX` function call or as `-1` for the COE default printer.

The DII COE Printer API also consists of six executable programs. The first three executable programs provide the same functionality as the C functions of the same name.

- C `EM_get_current_printer_name`
- C `EM_get_current_printer_type`
- C `EM_get_current_printer_desc`
- C `COEPrtBanner`
- C `COEPrtInstallDriver`
- C `COEPrtRemoveDriver`

Sample printer programs for the first three executable programs are shown below.

Sample Printer Programs

```
#include <stdio.h>

#include <Printer/PrintAPI.h>

/* Number of lines in text message */
#define TEXT_LINES  6

static char *PrintMessage[TEXT_LINES] =
{
  "Test message, line 1",
  "Test message, line 2",
  "Test message, line 3",
  "Test message, line 4",
  "Test message, line 5",
  NULL
};

int main(int argc, char *argv[])
{
  if (VPrintMsg(PrintMessage, TEXT_LINES) == -1)
    {
     fprintf(stderr, "Printer error.\n");
    }
}
```

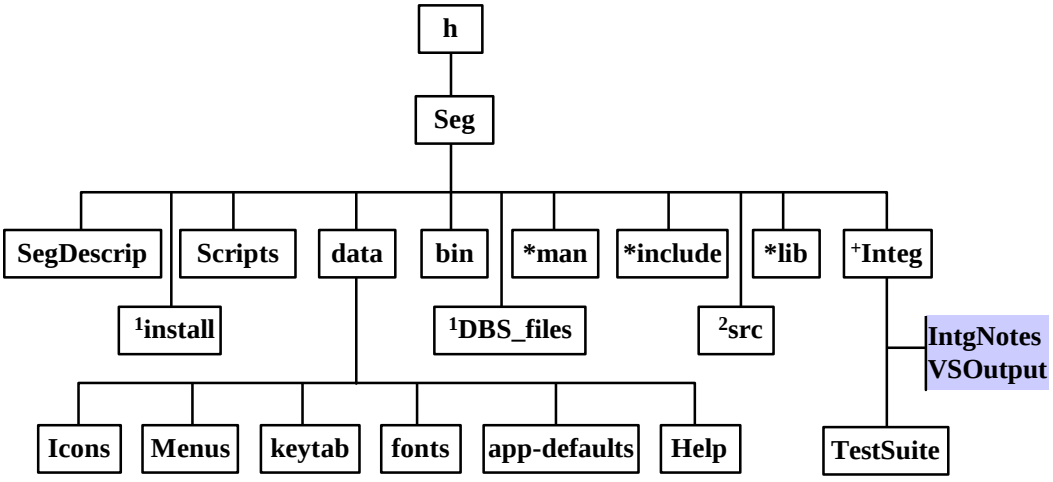
```
--------------------------------------------------------------------------------
#include <stdio.h>

#include <Printer/PrintAPI.h>

int main(int argc, char *argv[])
{
  int printer_num;

  /* Print a local text file using the system default printer */
  printer_num = VDirectPrintFile("textfile1", -1);
  if (printer_num == -1)

    {
      fprintf(stderr, "Printer error.\n");
      exit(1);
    }

  /* Print the system hosts file using a user-selected printer */
  printer_num = VPrintFile("/etc/hosts");
  if (printer_num == -1)
    {
      fprintf(stderr, "Printer error.\n");
      exit(2);
    }

  if (printer_num != -1)
    {
      /* Print another text file to the printer that the user just
         selected */
      printer_num = VDirectPrintFile("textfile2", printer_num);
      if (printer_num == -1)
        {
          fprintf(stderr, "Printer error.\n");
          exit(3);
        }
    }
}
```

4. Segment Development

The following section discusses the different types of segments and the process of segment creation. Refer to Section 5.0, *Runtime Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of segments.

4.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS (Commercial Off-the-Shelf), Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. For example, a `Scripts` subdirectory is required for an Account Group segment. The `Scripts` subdirectory normally contains scripts such as `.cshrc`, `.xsession`, and `.login`. These scripts serve as a template for establishing a basic runtime environment. For software segments, the `Scripts` subdirectory contains environmental extension files. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.



*** Required for segments with published APIs**
+ Required for segment submission
1 For Database segments only
2 Recommended location for source code during development, Required location for source code delivered to DISA.

Figure 1. Segment Directory Structure

The following runtime subdirectories normally are required, depending on the segment type: (1) `SegDescrip`, which is the directory containing segment descriptor files; (2) `Scripts`, which is the directory containing script files; (3) `bin`, which is the directory containing executable programs for the segment; and (4) `data`, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

The `SegDescrip` directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time, while APIs for the segments that own the data are used at runtime. Refer to Section 5.5, *Segment Descriptors,* of the *DII COE Integration and Runtime Specification* for a detailed explanation of `SegDescrip` files.

4.2 Running the COE Tools From the Command Line

The COE tools were constructed to aid developers in the creation and ultimate installation of DII COE segments. All tools can be run from the command line, and some can be run from other code using published APIs.

This section provides a brief overview of running the COE tools from the command line. When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools are used to communicate with the outside world in two ways. First, the tools use the exit function to set the UNIX `status` environment variable. The `status` return value is set to `0` for normal tool completion or to `-1` if an error occurs. A `status` return value greater than `0` indicates a completion code that is tool specific.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs.

NOTE: Redirecting `stdin` is not always convenient. The `-R` command line parameter allows a tool to read input from a response file instead of from `stdin`.

For example, the COEPrompt tool displays a message and allows the user to type a response. The user's response, then, is written to `stdout`. The following statement shows how this tool can be used to ask the user to enter the name of a file:

```
COEPrompt "Enter Filename" | MyProg
```

Or, the following statement can be used to write the results to a file:

```
COEPrompt "Enter Filename" > /tmp/tempfile
```

4.2.1 COE Runtime Tools

This section lists the COE tools that are available at runtime. These executables are delivered to the operational site and are located underneath the `h/COE/bin` directory.

4.2.1.1 COEAskUser

USAGE

```
COEAskUser [flags] "<msg>"
```

Where "`<msg>`" is the prompt to display to the user.

USABLE FLAGS

`-h, -H`	Display this help message.
`-AC`	Display buttons labeled as `Accept` and `Cancel`.
`-B b1 b2`	Display buttons labeled as `b1` and `b2`.
`-C <file>`	Read command line arguments from the named <`file`>.
`-TF`	Display buttons labeled as `True` and `False`.
`-V`	Display the tool's version number.
`-YN`	Display buttons labeled as `Yes` and `No`. This is the default if no button option is named.

This tool is intended for use in the `PostInstall` script to display a message to the user and to have the user click on a `Yes` or `No` button. The parameters allow the calling routine to specify the labels to be used for the `Yes` or `No` buttons. The `status` environment variable is set to `0` if the user clicked on the equivalent of the `No` button, or to `1` if the user clicked on the equivalent of the `Yes` button. The `status` environment variable is set to `-1` if an error occurred (for example, the requested message could not be displayed).

4.2.1.2 COEFindSeg

USAGE

```
COEFindSeg [flags]
```

USABLE FLAGS

`-h, -H`	Display this help message.
`-C file`	Read command line arguments from the named <`file`>.
`-d <dirname>`	Use <`dirname`> as the name of the directory where the segment is assumed to be located.
`-n <segname>`	Use <`segname`> as the name of the segment to locate as specified in the segment's `SegName` descriptor.
`-p <path>`	Use <`path`> as the absolute path to the segment.
`-V`	Display the tool's version number.

This tool returns information about a requested segment. The tool accepts as input the name of the directory where the segment is expected to be located, the segment name, and the segment prefix. If the directory is omitted or the expected directory is not present, the tool searches for the segment in the legal directory locations for segments.

The tool returns the absolute path of the directory where the segment was found, the segment name found, and the segment type, including the segment attribute if applicable. The tool returns the following information if it is successful

```
<directory> : <segname> : <prefix> : <type> [:attrib]
```

where <`directory`> is the absolute path name where the segment was found, <`segname`> is the name of the segment as indicated in the segment's `SegName` file, <`prefix`> is the segment prefix as indicated in the segment's `SegName` file, `type` is the segment type (translated to all uppercase), and `[attrib]` is the attribute, if any, found in the segment's `SegName` descriptor.

4.2.1.3 COEInstaller

USAGE

```
COEInstaller [flags]
```

USABLE FLAGS

`-h, -H`	Display this help message.
`-C <file>`	Read command line arguments from the named <`file`>.
`-d`	Set the debug flag.
`-s`	Run in installation server mode.
`-v`	Show verbose messages while the tool runs.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.

This tool displays a list of configuration definitions or segments that may be installed from tape or disk (for example, a network segment server). The `status` environment variable is set to `0` if all requested segments were installed correctly or to `-1` if any segment requested was not installed. By default, this tool does not write any output to `stdout`. This tool writes information to a status log that indicates installation progress, which segments have been installed, and other information that might be useful to the site administrator.

4.2.1.4 COEInstError

USAGE

```
COEInstError [flags] “<errmsg>”
```

Where `"<errmsg>"` is a string to display.

USABLE FLAGS

`-h, -H`	Display this help message.
`-V`	Display the tool’s version number.

This tool is intended to be used by a `PreInstall`, `PostInstall`, or `DEINSTALL` script to display an error message to the user. A message is displayed in a window and the user must click on an `OK` button to continue. This tool *always* returns a non-zero value.

4.2.1.5 COEMsg

USAGE

```
COEMsg [flags] "<msg>"
```

Where "<msg>" is the string to display.

USABLE FLAGS

-h, -H	Display this help message.
-V	Display the tool's version number.

This tool is intended to be used by a PreInstall, PostInstall, or DEINSTALL script to display an informational message to the user. A message is displayed in a window and the user must click on an OK button to continue. The status environment variable is set to -1 if an error occurs (for example, a window could not be displayed) or to 0 if the message displays correctly.

4.2.1.6 COEPrompt

USAGE

```
COEPrompt [flags] "<msg>"
```

Where "<msg>" is the string to display.

USABLE FLAGS

-h, -H	Display this help message.
-C <width>	Use <width> as the maximum number of characters the user can type in the response (the default is 40 characters).
-V	Display the tool's version number.

This tool is intended to be used by a PreInstall and PostInstall script to display an informational message to the user. The user must enter a response to the message. The calling routine may indicate the maximum number of characters in the user's typed response. The default, if not specified, is 40 characters. A prompt is displayed in a window, and the user must respond to the prompt and then click on the OK button to continue. The status environment variable is set to -1 if an error occurs (for example, a window could not be displayed) or to 0 if the prompt displays correctly. The user's response is written as a character string to stdout in the form

```
n
string
```

where n is the number of characters in the string that follows. If n is 0, the user entered a null response and no string follows.

4.2.1.7 COEPromptPasswd

USAGE

```
COEPromptPasswd [flags] "<msg>"
```

Where "<msg>" is the string to display.

USABLE FLAGS

-h, -H	Display this help message.
-C <width>	Use <width> as the maximum number of characters the user can type as a password (the default is 40 characters).
-n	Do not display the verify prompt.
-V	Display the tool's version number.

This tool is intended to be used in a PreInstall and PostInstall script to prompt a user to enter a password. The routine displays a window in which the user may enter a password. The user's response is not echoed in the window. A password prompt is displayed in a window, and the user must respond to the enter a password and verify the password. The user must then click on the OK button to continue. The status environment variable is set to -1 if an error occurs (for example, a window could not be displayed) or to 0 if the prompt displays correctly.

By default, the tool will prompt for the password and then prompt the user to enter the password again for confirmation. A flag may be passed to the tool to indicate that a confirmation prompt is not desired. The default maximum password length that a user can enter is 40 characters, while the minimum is 6 characters. The calling routine may indicate the minimum and maximum number of characters in the user's typed response.

When confirmation is requested, the tool prompts the user for a matching confirmation up to three times before returning a failure. If after three attempts the confirmation does not match, the tool returns an error code and the calling routine must determine what action to take.

4.2.1.8 COEUpdateHome

USAGE

```
COEUpdateHome [flags] <scriptname> <envvar>
```

Where `<scriptname>` is the filename, including the subdirectory, where the script file exists relative to the home directory, and `<envvar>` is the environment variable to update.

USABLE FLAGS

`-h, -H`	Display this help message.
`-V`	Display the tool's version number.

This tool is intended to be invoked from within a segment's `PostInstall` script. Its purpose is to update the home environment variable within a script file to point to where a segment was actually installed. This tool searches the named script file for the first place that the environment variable is defined through either `set` or `setenv` and replaces the definition with the absolute pathname for where the segment was loaded.

For example, assume the COEInstaller tool loads `MySeg` underneath `/home4/MySeg` and the script file `.cshrc.MYSEG` contains the following:

```
setenv MYS_HOME      /h/MySeg
```

Then the statement

```
COEUpdateHome Scripts/cshrc.MYSEG MYS_HOME
```

changes the statement to read

```
setenv MYS_HOME      /home4/MySeg
```

The `status` environment variable is set to `0` if the tool is successful, to `-1` if the file specified is not found, or to `1` if the file is found but the environment variable specified is not found.

4.2.2 COE Developer's Tools

This section lists the COE tools that are available during development but that are not delivered to operational sites. By default, these executables are located underneath the `DII_DEV` directory and are distributed as part of the Developer's Toolkit. These tools are not location sensitive and may be moved to any directory desired for development. For example, if the toolkit is tarred to `/h`, the path would be `/h/DII_DEV`.

All of the tools in this section are executed from a command line. None of these tools have a GUI interface.

4.2.2.1 CalcSpace

USAGE
`CalcSpace [flags] <segdir>`

Where `<segdir>` is the home directory of the segment.

USABLE FLAGS

`-h, -H`	Display this help message.
`-p <path>`	Use `<path>` as the absolute path to the segment.
`-v`	Show verbose messages while the tool runs and cause the tool to print the space requirements for each top-level directory.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.

This tool computes the amount of space (in K bytes) that a segment is using. The segment should not be compressed in any way. The calculated value is written in the `Hardware` descriptor. Warnings will be issued for directories that are found but not expected (for example, `include`, `src`) or if expected directories are missing (for example, `bin` for a software segment). The total space calculated will be printed to `stdout`. The reserve space portion of the `Hardware` descriptor is not affected nor are any specified partitions. The `status` environment variable is set to `0` upon completion or to `-1` if the `segdir` specified does not exist or does not appear to be a segment.

NOTE
`<path>` length and `<segdir>` length must be <= 256.

NOTE
If no path is specified, `/h` will be used.

4.2.2.2 CanInstall

USAGE
```
CanInstall [flags] <segdir>
```

Where <segdir> is the home directory of the segment.

USABLE FLAGS

-h, -H	Display this help message.
-p <path>	Use <path> as the absolute path to the segment.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool tests a segment to see if it can be installed. It performs the same tests that the COEInstaller tool performs at installation time. To be installable, all required segments already must be on the disk. In addition, the disk must not contain any conflicting segments. The `status` environment variable is set to `0` if the segment can be installed or to `-1` if the segment cannot be installed. An error message is printed to indicate why a segment cannot be installed.

NOTE
If no path is specified, `/h` will be used.

4.2.2.3 ConfigDef

USAGE
`ConfigDef {[general flags] [specific flags]}`

USABLE FLAGS

`-h,-H:`	Display this help message.
`-f:`	Print segment names, segment directories as they are processed. If they are available.
`-p <path>`	Use `<path>` as the absolute path to the segment.
`-v`	Show verbose messages while the tool runs.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.
`-a <file>:`	Use `<file>` as the name of distribution file to append.
`-o <file>:`	Use `<file>` as the name of distribution file.
`-nb bundle [:file]:`	Use `[:file]:` to create a bundle definition or reference.
`-nc name [:file]:`	Use `[:file]:` to create a configuration definition or reference.
`-ns <segdir>:"seg"[:ver]` OR `<segdir>[::ver]:`	Create a segment reference.
`-r <"text">`	Use `<"text">` to specify quoted release note text for a bundle or configuration.
`-rf <file>`	Use `<file>` to specify a release note file for a bundle or configuration.
`-s <segdir>:"seg"[:ver]` OR `<segdir>[::ver]:`	Create a segment reference within a bundle or configuration.
`-vl:`	Validate the distribution definition.
`-vr <version>:`	Use `<version>` to specify a version for a bundle or configuration.

This tool is used to create a configuration definition from a list of segments. Components of a configuration definition (e.g., bundles, configurations, folders) may participate in more than one configuration definition.

Segments will normally reside in a segment repository such as SDMS. Thus, this tool provides an interface that allows segments to be checked out of the repository. The interface is the same as that provided by MakeInstall.

NOTE
If no path is specified, `/h` will be used.

4.2.2.4 ConvertSeg

USAGE
`ConvertSeg [flags] <segdir> ...`

Where <segdir> is the home directory of the segment to be converted.

USABLE FLAGS

-h, -H	Display this help message.
-p <path>	Use <path> to establish a path for subsequent file names.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.

This tool examines segment descriptors and converts them to the latest format. To the extent possible, obsolete usage is identified and corrected as part of this process. The original segment descriptor directory is not modified, but it is renamed to be `SegDescrip.orig`. The converted segment descriptors created by the tool are placed in a newly created `SegDescrip` directory (for example, after the original `SegDescrip` is moved to `SegDescrip.orig`). To avoid inadvertently overwriting a segment descriptor directory, a prompt is issued if the directory `SegDescrip.orig` already exists. ConvertSeg should be run as the `root` user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE
The `-p` flag and <segdir> can be specified more than once on the command line.

NOTE
If no path is specified, `/h` will be used.

4.2.2.5 MakeAttribs

USAGE
`MakeAttribs [flags] <segname>`

USABLE FLAGS

`-h, -H`	Display this help message.
`-p path`	Use <`path`> to establish a path for subsequent file names.
`-v`	Show verbose messages while the tool runs.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.

This tool creates the descriptor file `FileAttribs`. It traverses every subdirectory beneath the segment's home directory and creates a file with lines in the following format:

```
<permits> : <owner> : <group> : <filename>
```

At installation time, the installation tools perform the following statements for each entry:

```
chmod permits $INSTALL_DIR/filename
chown owner $INSTALL_DIR/filename
chgrp group $INSTALL_DIR/filename
```

For security reasons, MakeAttribs does *not* include any file owned by `root` or any file for which *permits* is greater than 777. A warning is printed for each filename that is rejected. A warning is also printed for each "suspicious" permission such as 777 for a file, execute permissions on files in a `data` subdirectory, and so forth. This tool must be run as the `root` user because it modifies files the user may not own.

NOTE
If no path is specified, `/h` will be used.

4.2.2.6 MakeInstall

USAGE

```
MakeInstall [flags] <segname>
```

Where <segname> is a list of one or more segments to process.

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-Cd <exe> <path>	Use <exe> as the name of the executable that will check the segment descriptor out of the repository and place it in the temporary location indicated by absolute path name <path>. The location of the executable is determined by the most recent occurrence of the -p flag.
-Co <exe> <path>	Use <exe> as the name of the executable that will check the segment out of the repository and place it in the temporary location indicated by the absolute path name <path>. The location of the executable is determined by the most recent occurrence of the -p flag.
-d on | off	Specify whether to delete (-d on) or not to delete (-d off) segments checked out of the repository when MakeInstall is completed.
-di <definitions>	Use <definitions> as a list of one or more distribution definition files as created by ConfigDef. The location of the distribution is determined by the most recent occurrence of the -p flag. These distributions are not in a repository. All of the segments that comprise the distribution must be located by the -p flag or through the directory list in the $SEG_DIRS path environment variable or else an error will be generated.
-dio <definitions>	Use <definitions> as a list of one or more distribution definition files as created by ConfigDef. The definitions are to be checked out of the repository by the program provided by the -Co and -Cd flags. The segments that comprise the distribution also must be included (via -s, -so, -S, or -So) or an error will be generated.
-Di <files>	Use <files> as a list of one or more files that contain a list of distribution definitions to write to the output medium. The location of the file is determined by the most recent occurrence of the -p flag, while the distribution definitions are located by the $SEG_DIRS environment variable or by -p flags in the list file. These distributions are not in a repository. All of the segments that comprise the distribution must be located by the -p flag or through the directory list in the $SEG_DIRS path environment variable or else an error will be generated.
-Dio <files>	Use <files> as a list of one or more files that contain a list of distribution definitions to write to the output medium. The list of distribution definitions are to be checked out of the repository specified by the -Co and -Cd flags. All of the segments that comprise the variant also must be included (via -s, -so, -S, or -So) or an error will be generated.
-f	Echo segment names as they are processed (default is OFF).
-o <file>	Use the disk as the output medium instead of tape. The <file> created is in tar format and has the filename file.tar.

ot <files>	Use <files> as a list of one or more segment files created by MakeInstall with the -o flag. This flag writes files to the output medium in a format suitable for installation by the COEInstaller tool.
-p <path>	Use <path> to establish a path to segments, variants, and so forth.
-R <file>	Use <file> to respond to questions from the tool.
-s <segs>	Use <segs> as a list of one or more segments to write to the output medium. The location of the segments is determined by most recent occurrence of the -p flag. These segments are not in a repository.
-so <segs>	Use <segs> as is a list of one or more segments to write to the output medium. The segments are to be checked out of the repository by the program provided by the -Co and -Cd flags.
-S <files>	Use <files> as a list of one or more files that contain a list of segments to write to the output medium. The location of the files is determined by the most recent occurrence of the -p flag. Segments are located under /h or as determined by the -p flag within the file. These segments are not in a repository.
-So <files>	Use <files> as a list of one or more files that contain a list of segments to be checked out of the repository specified by the -Co and -Cd flags.
-t <dev>	Use <dev> as the output device (for example, /dev/rmt/3m).
-T	Read and display the tape's table of contents.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.
-x	Validate but do not actually create a tape.

This tool writes one or more segments to an installation medium and packages the segments for distribution over the network. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. The `status` environment variable is set to `-1` if any errors occur and to `0` if the process is successful. Error messages are written to `stdout` as appropriate.

NOTE
The flags `-Cd`, `-Co`, `-dio`, `-Dio`, `-ot`, `-R`, `-so`, `-So`, and `-T` are not currently supported.

NOTE
If no path is specified, `/h` will be used.

4.2.2.7 TestInstall

USAGE
`TestInstall [flags] <segment_list>`

USABLE FLAGS

`-h, -H`	Display this help message.
`-C <file>`	Read command line arguments from the named `<file>`.
`-e`	Echo descriptor contents as they are processed. Enabling this flag thereby enables the `-f` flag.
`-f`	Echo descriptor names as they are processed.
`-p <path>`	Use `<path>` as the absolute path to the source directory.
`-v`	Show verbose messages while the tool runs.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.

This tool is used to temporarily install a segment that already resides on disk. It must be run when no other COE processes are running. The reason for this restriction is that the tool may modify COE files already in use with unpredictable results. VerifySeg must have been run before TestInstall to make sure the segment is valid.

This tool performs the same operations as the COEInstaller tool except that it does not need to read the segment from tape (for example, it is already on disk), and the segment may be in any arbitrary location. This tool will establish the required symbolic link under `/h` to preserve the COE standard directory structure. The `status` environment variable is set to `0` if the installation is successful or to `-1` if the installation is not successful. Error and status messages are written to `stdout` as required. The TestInstall tool must be run as the `root` user because it modifies files the user may not own.

NOTE
If no path is specified, `/h` will be used.

4.2.2.8 TestRemove

USAGE
`TestRemove [flags] <segname>`

Where `<segname>` is the name of the segment to be removed.

USABLE FLAGS

`-h, -H`	Display this help message.
`-C <file>`	Read command line arguments from the named `<file>`.
`-e`	Echo descriptor contents as they are processed--enabling this flag thereby enables the `-f` flag.
`-f`	Echo descriptor names as they are processed.
`-p <path>`	Use `<path>` as the source directory.
`-v`	Show verbose messages while the tool runs.
`-V`	Display the tool's version number.
`-w`	Suppress all warnings.

This tool is used to remove a segment that was installed by TestInstall. It must be run when no other COE processes are running. The reason for this restriction is that the tool may modify COE files already in use with unpredictable results. This tool removes the symbolic link under `/h` if one exists, but it does *not* delete the segment from disk. The `status` environment variable is set to `0` if the removal is successful or to `-1` if the removal is not successful. Error and status messages are written to `stdout` as required. The TestRemove tool must be run as the `root` user because it modifies files the user may not own.

NOTE
This tool is unconditional and should be used with great caution. It will remove a specified segment even if other installed segments still depend on it.

NOTE
If no path is specified, `/h` will be used.

4.2.2.9 TimeStamp

USAGE
`TimeStamp [flags] <segname>`

USABLE FLAGS

`-h, -H`	Display this help message.
`-p <path>`	Use <`path`> to establish a path for subsequent file names.
`-V`	Display the tool's version number.

This tool puts the current time and date into the `VERSION` segment descriptor. It is intended to assist the configuration process by allowing the time stamp to be updated just before running VerifySeg and mkSubmitTar for the deliverable product. The `status` environment variable is set to `-1` if an error occurs (for example, the `VERSION` segment descriptor is not found) or to `0` if the time stamp is successful. TimeStamp should be run as the `root` user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE
If no path is specified, `/h` will be used.

4.2.2.10 VerifySeg

USAGE
`VerifySeg [flags] <segdir> ...`

Where <segdir> is the home directory of the segment to be verified.

USABLE FLAGS

-h, -H	Display this help message.
-C <file>	Read command line arguments from the named <file>.
-e	Echo descriptor lines as they are processed.
-f	Echo descriptor names as they are processed.
-o	Identify obsolete usage in the segment.
-p <path>	Use <path> to establish a path for subsequent file names.
-R <file>	Use responses listed in <file> to answer questions.
-s <name>	Only validate the descriptor <name>.
-t	Print a table of required/optional descriptors.
-v	Show verbose messages while the tool runs.
-V	Display the tool's version number.
-w	Suppress all warnings.
-x <name>	Display syntax for the descriptor <name>.

This tool is used to validate that a segment conforms to the rules for defining a segment. It uses information in the SegDescrip subdirectory and must be run whenever the segment is modified. This tool must be run for each segment. If the segment is an aggregate segment, it must be run on each segment in the aggregate. The status environment variable is set to -1 if any errors occur or to 0 if the validation is successful. Error and status messages are written to stdout as required. VerifySeg should be run as the root user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE
The -p flag and <SegDir> can be specified more than once on the command line.

NOTE
If no path is specified, /h will be used.

4.2.2.11 VerUpdate

USAGE
`VerUpdate [flags] <segname>`

USABLE FLAGS

`-h, -H`	Display this help message.
`-V`	Display the tool's version number.
`-p <path>`	Use `<path>` to establish a path for subsequent file names.
`-i <version>`	Use `<version>` as the version number to insert unconditionally.
`-d <digit>`	Use `<digit>` to increment the digit specified by one in the version number. For example, the version number `<1.2.3.4>`. `<digit>` is 1 for the major release digit `<digit>` is 2 for the minor release digit `<digit>` is 3 for the maintenance release digit `<digit>` is 4 for the developer digit
`-ip <version>`	Use `<version>` as the patch version number to insert unconditionally.
`-ap`	Add path version number '`P1`' to the `VERSION` file if it does not exist already.
`-up`	Increment the patch version number by one if it exists.

This tool updates the segment version number, current time, and current date in the `VERSION` descriptor. It is intended to assist the configuration process by allowing the version, current time, and current date to be updated just before running VerifySeg and mkSubmitTar for the deliverable product. The `status` environment variable is set to `0` if the version update is successful or to `-1` if an error occurs (for example, the segment was not found). VerUpdate should be run as the `root` user, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed.

NOTE
If no path is specified, `/h` will be used.

NOTE
Command line parameters `-ip`, `-ap`, and `-up` may only be used one at a time.

4.3 Building Your Segment

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, `SegDescrip`, which is the segment descriptor subdirectory.

This section describes a process to turn an application into a segment so it can be a part of the DII COE. As described earlier, a segment is a collection of one or more CSCIs most conveniently managed as a unit.

4.3.1 Identifying and Creating Required Subdirectories

There are six segment types: Account Group, COTS, Data, Database, Software, and Patch. Each segment type is assigned its own subdirectory. Precise files depend on the segment type.

The following subdirectories normally are required:

Subdirectory	Description
`SegDescrip`	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to modify any files directly for resources it does not *own*; in other words, a segment cannot modify files or resources outside an assigned directory. The DII COE tools coordinate the modification of all community files at installation time, while APIs for the segment that owns the data are used at runtime. This subdirectory contains the installation instructions for the segment.
`Scripts`	Subdirectory containing script files. This subdirectory normally will contain scripts such as `.cshrc`, `.xsession`, and `.login`. These scripts serve as a template for establishing a runtime environment.
`bin`	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of segment.
`data`	Subdirectory for static data items, such as menu items, that are unique to the segment but that will be the same for all users on all workstations.

Reference Sections 5.0-5.5 of the *DII COE Integration and Runtime Specification* for a detailed explanation of segment directory layout and a description of each `SegDescrip` file.

4.3.2 Creating or Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Reference Table 1 to determine the descriptor files required for each segment type. For example, the AcctGrp segment requires `ReleaseNotes`, `SegInfo`, `SegName`, and `VERSION` descriptor files in the `SegDescrip` directory, while the Patch segment requires the `PostInstall` descriptor in addition to the previously listed files. Some segment descriptor information is provided within the files listed in Table 1.

NOTE: In Table 1, `Aggregate` and `COE Comp` are segment attributes that can be associated with any type of segment.

File	Acct Grp	COTS	Data	DB	S/W	Patch
DEINSTALL	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O
Installed	I	I	I	I	I	I
PostInstall	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R
SegName	R	R	R	R	R	R
Validated	I	I	I	I	I	I
VERSION	R	R	R	R	R	R
R - Required O - Optional I - Created by Integrator or Installation Software						

Table 1. Segment Descriptor Files

Other segment descriptor information is arranged within subsections of the `SegInfo` file. As with the descriptor files themselves, some sections of the `SegInfo` file are required and others are optional depending on the type of segment. Table 2 defines the required and optional sections for each segment type.

Section	Acct Grp	COTS	Data	DB	S/W	Patch
AcctGroup	R	N	N	N	N	N
COEServices	O	O	O	O	O	O
Community	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O
Compat	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O
Data	N	N	R	N	N	N
Database	N	N	N	R	O	O
DCEDescrip	N	O	N	N	O	N
Direct	O	O	O	O	O	O
FilesList	O	R	O	O	O	O
Hardware	R	R	R	R	R	R
Help	O	O	O	O	O	O
Icons	R	O	N	N	O	O
Menus	R	O	N	N	O	O
Network	N	N	N	N	N	N
Permissions	O	N	N	N	O	O
Processes	O	O	N	N	O	O
ReqrdScripts	R	N	N	N	O	N
Requires	O	O	O	O	O	O
Security	R	R	R	R	R	R
SharedFile	O	O	N	N	O	O
R - Required		O - Optional		N - Not Applicable		

Table 2. SegInfo Segment Descriptor Sections

4.3.3 Installing a Segment

Follow the procedures below to install a segment after it has been created.

Run VerifySeg

The VerifySeg tool must be run during the development phase to ensure segments use segment descriptor files properly. Run the VerifySeg tool whenever a segment is created or modified. When VerifySeg is run to verify a segment, a `Validated` file is created. This file is required to create the installation media or to use the TestInstall tool on the segment. Reference Section 4.2.2.11, *VerifySeg*, for further information about using VerifySeg.

Run TestInstall

Executing the TestInstall tool is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation of a segment on the developer's workstation before actual installation. Reference Section 4.2.2.8, *TestInstall*, for further information about using TestInstall.

Run MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment(s) for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. Reference Section 4.2.2.6, *MakeInstall*, for further information about using MakeInstall.

Run COEInstaller

The COEInstaller tool installs a segment from tape, disk, or other electronic media. Reference Section 4.2.1.3, *COEInstaller*, for further information about using COEInstaller.

4.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, although the segments may need to add menu items and icons. Some segments may need to add special extensions such as sockets. This subsection describes how to add menu items, icons, and special extensions.

4.4.1 Adding Menu Items

Menu Entry Format

The Menu Descriptor in the `SegInfo` file is used to specify the name of the segment's menu file and the name of the affected segment's menu file.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the entries in the named menu file. The format of the entries is in ASCII with colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. You may use any or all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs:

A **`PDMENU`** line contains the following elements:

```
PDMENU: name : enable flag : id # :
```

`PDMENU`	Keyword that indicates the start of a pull-down menu.
`name`	Text used to name the menu. The menu name is displayed on the menu bar.
`enable flag`	Integer value that indicates whether a menu is enabled or disabled. The enable flag is `1` if a menu is enabled or `0` if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.
`id#`	Optional integer value that provides a unique ID number for the menu. The `PDMENU id#` value must be unique within the menu description file. An absolute value may be provided. However, the `id#` field should be left empty so that relative numbering is used by default. With relative numbering, an `id#` of `R1` (or leaving the field blank) sets the menu's ID number to `1` plus the `id#` of the last menu processed. An `id#` of `R2` sets the menu's ID number to `2` plus the `id#` of the last menu processed.

The following is an example of a `PDMENU` line:

```
PDMENU: Map Options : 1 : R1 :
```

A **PDMENUEND** line contains the following element:

```
PDMENUEND:
```

PDMENUEND	Optional keyword that indicates the end of a group of pull-down menu items. If PDMENUEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of a PDMENUEND line:

```
PDMENUEND:
```

An **ITEM** line contains the following elements:

```
ITEM: name : command : execution type : enable flag : # instances : id# :
check value : security char : autolog flag : print flag : disk flag :
```

ITEM	Keyword that indicates a menu item description line.
name	Text used to name the menu item. The item name is displayed in the pull-down menu.
command	Program with space-separated arguments that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.
execution type	Integer value that indicates how to execute a command, as follows: 1 = executable program 2 = void callback function with no parameters (not yet implemented) 3 = Motif callback function (not yet implemented).
enable flag	Integer value that indicates if a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.
# instances	Integer value that is used to set the maximum number of times the item can be executed simultaneously.
id#	Optional integer value that provides a unique ID number for the menu item. Each ITEM id# entry must be unique within a PDMENU listing. (ITEM entries in a PRMENU must be unique within that PRMENU.) Refer to the id# description under the PDMENU keyword listing.

check value Optional integer value that sets the star and check annotations of a menu item. The possible values are:

```
0 = no annotation (default)
1 = visible check mark
2 = check mark, but not visible
3 = visible star
4 = star member, but not visible.
```

This element is not yet fully implemented.

security char Optional character value that is used to determine the lowest security level under which a menu item can be classified. Valid settings are:

```
N = No Classification
U = Unclassified (default)
C = Confidential
S = Secret
T = Top Secret.
```

autolog flag Optional character value, `T` or `F`, used to indicate if the command should be logged automatically. This element is not yet fully implemented.

print flag Optional character value, `T` or `F`, used to indicate if the command should have a print capability. This element is not yet fully implemented.

disk flag Optional character value, `T` or `F`, used to indicate if the command should have disk access capability. This element is not yet fully implemented.

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **`PRMENU`** line contains the following elements:

```
PRMENU: name : enable flag : id# :
```

`PRMENU` Keyword that indicates a cascading menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.

name Text used to name the cascade menu with which to connect. The `PRMENU` name is displayed in the pull-down menu.

enable flag Integer value that indicates if a cascade menu is enabled or disabled. The enable flag is `1` if a cascade menu is enabled or `0` if it is disabled. A disabled cascade menu means that menu options on the cascade menu cannot be selected.

id# Optional integer value that provides a unique ID number for the cascading menu. Each `PRMENU id#` must be unique within a `PDMENU` listing. Refer to the `id#` entry under the `PDMENU` keyword listing.

The following is an example of a `PRMENU` line:

```
PRMENU: Software : 1 : R1 :
```

A **CASCADE** line contains the following element:

```
CASCADE: name :
```

CASCADE	Keyword that indicates the start of a cascade menu. The cascade menu connects to the PRMENU entry of the same name.
name	Text used to name a cascade menu. The name is used to attach a cascade menu to a cascading button. This name must be the same as the name field in the PRMENU entry.

The following is an example of a CASCADE line:

```
CASCADE: Software :
```

A **CASCADEEND** line contains the following element:

```
CASCADEEND:
```

CASCADEEND	Optional keyword that indicates the end of a group of cascade menu items. If CASCADEEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of a CASCADEEND line:

```
CASCADEEND:
```

An **APPEND** line contains the following elements:

```
APPEND: name :
```

APPEND	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not exist already. The group is appended to the PDMENU or CASCADE entry of the same name.
name	Text used to select the menu to which a group of items is appended.

The following is an example of an APPEND line:

```
APPEND: Options :
```

An **APPENDEND** line contains the following element:

```
APPENDEND:
```

APPENDEND	Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If APPENDEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of an APPENDEND line:

```
APPENDEND:
```

A **SEPARATOR** line contains the following element:

```
SEPARATOR:
```

SEPARATOR	Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a SEPARATOR line:

```
SEPARATOR:
```

Example of Adding a Menu Item

To add menu items, include the Menus Descriptor in the SegInfo Segment Descriptor file. Specify the Menu file you use wish to load and the Menu file you wish to update. The Menu file you wish to load should be located in the Menus directory of the segment. If your segment name is TstSeg, the file would be located in the TstSeg/data/Menus directory. The following example will add the Test Program menu item to the Software menu under the SysAdm account group by updating the SA_Default.main menu file.

The following file changes must be made to ensure the TSTCOEAskUser_example program is executed from the Software menu, Test Program option:

TstSeg/SegDescrip/SegInfo entry:

```
[Menus]
TstSegMenu:SA_Default.main
```

TstSeg/data/Menus/TstSegMenu entry:

```
#--------------------
# Software Menu Items
#--------------------
APPEND          :Software
ITEM            :Test Program   :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

The $SEGMENT keyword must be used in the SegName Segment Descriptor file to specify the name of the affected segment. In this case it is System Administration.

```
#
# SegName For the TstSeg segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

4.4.2 Adding Icons

Icon Entry Format

The Icon Description Entry contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the CDE. The entry must be available to the CDE at startup as part of the base set of icons.

Icons are built using the icon section in the SegInfo file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A # symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```

The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Window Title : Icon Path : Executable Name : Comments
```

(where Window Title is the title placed in the application window, Icon Path is the full path to the pixmap/xpm image, Executable Name is the name of the executable to be launched by the menu program, and Comments is an optional comment line)

An example of an affected icon file is as follows:

```
Edit Profiles : /h/AcctGrps/SysAdm/data/Icons/Prof.img:EditProfiles : This
is the EditProfiles icon
```

Example of Adding an Icon

To add an icon, include the `Icons` Descriptor in the `SegInfo` Segment Descriptor file. Specify the icon file you wish to load and the icon file you wish to update. The icon file you wish to load should be located under the `TstSeg/data/Icons` directory, assuming the segment's directory name is `TstSeg`. This example will add the `Test Program` icon to the SysAdm account group. When invoked through the icon, the program `TSTCOEAskUser_example` will be executed.

`TstSeg/SegDescrip/SegInfo entry:`

```
[Icons]
TstSegIcons:SA_Default
```

`TstSeg/data/Icons/TstSegIcon file:`

```
TstSegIcons
#--------------------
# Software Icons
#--------------------
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

The `$SEGMENT` keyword must be used in the `SegName` Segment Descriptor file to specify the name of the affected segment. In this case it is `System Administration`.

```
SegName

#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

4.4.3 Reserving a Socket

To add a service, include the `COEServices` Descriptor in the `SegInfo` Segment Descriptor file. Also include the `$SERVICES` keyword in the `SegInfo` Segment Descriptor file to specify the service to be added. If the port number requested is already in use under another name, an error will be generated.

NOTE: Port numbers in the range 2000-2999 are reserved for DII COE segments.

```
[COEServices]
#
#        This is my service to add
#
$SERVICES
irc_ser:3001:upd
```

4.4.4 Displaying a Message

This subsection shows an example of how to display a message during the PostInstall process. Five runtime tools can be used to communicate with a user: COEAskUser, COEInstError, COEMsg, COEPrompt, and COEPromptPasswd. These tools may be used to display information to the user or to ask the user a question and, based on the result, perform different actions.

In this example, the user is asked questions using the COEAskUser runtime tool, which is described in Section 4.2.1.1, *COEAskUser*.

```
#!/bin/csh
#================================================================
# PostInstall                                    Tst 1.0 1/95
#
# Routine to perform necessary actions after TstSeg has been
# loaded.
#================================================================
COEAskUser -B "RED LAN" "BLUE LAN" "Which LAN Will You Be Connecting To"

if ($status == 1) then
  COEAskUser -YN "On The RED LAN Do You Want Port #66?
#
# Perform Some Action Based On Results
#
  exit(0)
else if ($status == 0) then
  COEAskUser -YN "On The BLUE LAN Do You Want Port #6?
#
# Perform Some Action Based On Results
#
  exit(0)
else
  COEMsg "Invalid Return Status"
  exit(-1)
endif
exit(0)
```

Appendix A - Sample Segment Layout

This appendix shows a layout of a software segment, called TstSeg, which is a basic example of a segment. In Appendix B, *Verifying Segment Syntax and Loading a Segment onto Tape*, TstSeg will pass the checks performed by the VerifySeg COE tool. TstSeg will add the `Test Program` menu item to the SysAdm account group. When invoked through the menu item, the `TSTCOEAskUser_example` program will be executed.

Refer to Appendix B, *Verifying Segment Syntax and Loading a Segment onto Tape*, for instructions on how to validate the `TstSeg` segment and load the segment onto tape.

The layout of the sample segment is:

```
TstSeg:
./       ../   Scripts/   SegDescrip/  bin/  data/

TstSeg/Scripts:
./       ../   .cshrc.TST

TstSeg/SegDescrip:
./       ../   DEINSTALL   ReleaseNotes   SegInfo  SegName   VERSION

TstSeg/bin:
./       ../   TSTCOEAskUser_example

TstSeg/data:
./       ../   Menus/

TstSeg/data/Menus:
./       ../   TstSegMenu
```

NOTE: After the segment has passed VerifySeg, a validated file will be added to the `SegDescrip` directory.

The `Scripts` directory contains the following:

.cshrc.TST

```
#===============================================================================
# Define required runtime environment variables
#===============================================================================
setenv TST_HOME     /h/TstSeg
#
# Add bin to path
#
set path=($path $TST_HOME/bin)
```

The `SegDescrip` directory contains the following:

DEINSTALL

```
# !/bin/csh
#
# Deinstall For TstSeg
#
```

NOTE: The presence of the `DEINSTALL` descriptor, even if it does not contain any instructions, allows the segment to be deinstalled.

ReleaseNotes

```
#
# Release Notes For TstSeg
#
This is my Test Segment For Example Purposes
```

SegInfo

```
#
# SegInfo File For TstSeg
#

[Hardware]
$CPU:HP
$MEMORY:200
$DISK:131

[Menus]
TstSegMenu:SA_Default.main

[Reqrd Scripts]
.cshrc:.cshrc.TST

[Security]
UNCLASS
```

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

TstSegMenu

```
#-------------------------
# Software Menu Items
#-------------------------
APPEND  :Software
ITEM  :Test Program  :TSTCOEAskUser_example:1:1:1:R1
APPENDEND:
```

VERSION

```
#
# Version Number For TstSeg
#
1.0.0.1 : 05/31/96: 10:08
```

Appendix B - Verifying Segment Syntax and Loading a Segment onto Tape

This appendix provides examples of how to convert a segment from the Joint Maritime Command Information System (JMCIS) format to the *DII COE Integration and Runtime Specification* segment format, verify segment syntax, install a segment temporarily, and load a segment onto an installation tape. The segment verification and loading process involves the following steps:

STEP 1: **Run the VerifySeg tool**. Run VerifySeg to validate that the segment conforms to the rules for defining a segment (that is, to verify the segment syntax).

STEP 2: **Run the TestInstall tool**. Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 3.

STEP 3: **Run the MakeInstall tool**. Run MakeInstall to load the segment onto an installation tape. After the segment is loaded onto tape, it is ready to be installed using the `Segment Installer` option from the System Administration menu bar.

Subsections B.1-B.3 show how to perform these steps against the TstSeg sample software segment, which is described in Appendix A, *Sample Segment Layout.*

NOTE: In the subsections below, the VerifySeg, TestInstall, and MakeInstall tools are being run against the TstSeg sample segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

`(F)`	indicates a FATAL ERROR	`(D)`	indicates a DEBUG statement
`(W)`	indicates a WARNING	`(V)`	indicates a VERBOSE statement
`(E)`	indicates an ERROR	`(O)`	indicates an ECHO statement

NOTE: In the following subsections, boldface text indicates information that the user must input.

B.1 Running VerifySeg Against the Sample Segment

```
*******************************************************************************

VerifySeg -p /home2/TestSegs TstSeg

Results of verification (/home2/TestSegs/TstSeg) :Totals
------

Errors:     0
Warnings:   0
*******************************************************************************
```

B.2 Running TestInstall Against the Sample Segment

```
*******************************************************************************

TestInstall -p /home2/TestSegs TstSeg

*******************************************************************************

TestInstall - Version 1.0.0.7

*******************************************************************************

The following options have been selected:

*******************************************************************************

Print warning messages.

*******************************************************************************

Segments to be TestInstalled:

*******************************************************************************

Segment: TstSeg Path: /home2/TestSegs

************WARNING*************

TestInstall may modify COE files already in use. This may cause unpredictable
results if COE processes are already running. Make sure no other COE processes
are running before using TestInstall.

Do you want to continue with the TestInstall? (y/n):y
Processing TstSeg
The segment /home2/TestSegs/TstSeg already uses the DII COE standard.
ConvertSeg is not required.

Successfully ran preprocessor on segment TstSeg

Do you want to run PreInstall for Segment TstSeg? (y/n):y
Calling PreInstall Script
effective 0 real 0

Do you want to run PostInstall for Segment TstSeg? (y/n):y
effective 0 real 0
Successful Installation of TstSeg

*******************************************************************************
```

B.3 Running MakeInstall Against the Sample Segment

```
*******************************************************************************
```

MakeInstall -p /home2/TestSegs TstSeg

```
1  Write to disk
2  /dev/rmt/3mn            3 MBytes (HP DAT DT-30)
3  /dev/rmt/3mn          680 MBytes (HP DAT DT-30)
4  /dev/rmt/3mn         1360 MBytes (HP DAT DT-60)
5  /dev/rmt/3mn         2048 MBytes (HP DAT DT-90)
6  /dev/rmt/3mn         2730 MBytes (HP DAT DT-120)
7  /dev/rmt/3mn         4096 MBytes (HP DAT DT-210)
8  /dev/rmt/stn           60 MBytes (HP 1/4 inch DC600 Cartridge)
9  /dev/rmt/stn          150 MBytes (HP 1/4 inch DC6150 Cartridge)
10 /dev/rmt/stn          250 MBytes (HP 1/4 inch DC6250 Cartridge)
11 /dev/rmt/stn          525 MBytes (HP 1/4 inch DC6525 Cartridge)
12 /dev/rmt/1mn         1225 MBytes (HP 54M Exabyte)
13 /dev/rmt/1mn         2560 MBytes (HP 112M Exabyte)
14 /dev/nrst0            680 MBytes (Sun DAT DT-30)
15 /dev/nrst0           1360 MBytes (Sun DAT DT-60)
16 /dev/nrst0           2048 MBytes (Sun DAT DT-90)
17 /dev/nrst0           2730 MBytes (Sun DAT DT-120)
18 /dev/nrst0           4096 MBytes (Sun DAT DT-210)
19 /dev/nrst0             60 MBytes (Sun 1/4 inch DC600 Cartridge)
20 /dev/nrst0            150 MBytes (Sun 1/4 inch DC6150 Cartridge)
21 /dev/nrst0            250 MBytes (Sun 1/4 inch DC6250 Cartridge)
22 /dev/nrst0            525 MBytes (Sun 1/4 inch DC6525 Cartridge)
23 /dev/nrst0           1225 MBytes (Sun 54M Exabyte)
24 /dev/nrst0           2560 MBytes (Sun 112M Exabyte)
25 /dev/nrtape           680 Mbytes (SGI DAT DT-30)
26 /dev/nrtape          1360 Mbytes (SGI DAT DT-60)
27 /dev/nrtape          2048 Mbytes (SGI DAT DT-90)
28 /dev/nrtape          2730 Mbytes (SGI DAT DT-120)
29 /dev/nrtape          4096 Mbytes (SGI DAT DT-210)
30 /dev/rmt0.1          1225 Mbytes (IBM 54M Exabyte)
31 /dev/rmt0.1          2560 Mbytes (IBM 112M Exabyte)
32 /dev/nrmt0h           680 Mbytes (DEC DAT DT-30)
33 /dev/nrmt0h          1360 Mbytes (DEC DAT DT-60)
34 /dev/nrmt0h          2048 Mbytes (DEC DAT DT-90)
35 /dev/nrmt0h          2730 Mbytes (DEC DAT DT-120)
36 /dev/nrmt0h          4096 Mbytes (DEC DAT DT-210)
37 Other

Enter device to use (1, 2, etc) or type 'q' to quit.1
Enter name of the output file or type 'q' to quit.TstSeg

Processing Segment: /home2/TestSegs/TstSeg ...

Enter your name for the Tape Header: John Smith
Enter a serial number for the Tape Header: 1
Enter any desired comment to put in the Tape Header (up to 255 characters):
Test Load of TstSeg
```

```
Tape Index Attr Type Hardware Class Directory (Segment Name - Version)
================================================================================
1      1    O    S      HP      U   /home2/TestSegs/TstSeg
                                     Test Segment - 1.0.0.1
================================================================================
Attr : PS - Parent Segment  CS - Child Segment  0 - Other
Type : A - Acct Group       S - Software        C - COTS
       D - Data             P - Patch
Class: U - Unclassified     C - Confidential    S - Secret  T - Top Secret

Number of segments to write to tape: 1
Space required: 0.114 MByte (including Tape Header and Table of Contents)

********************************************************************************

********* Insert tape #1 *********

Press any key to continue.
0+1 records in.
1+0 records out.

************ DII Install tape completed ************
```

Appendix C - Security Manager Configuration File

The file `/h/AcctGrps/SecAdm/data/config/secman_default`scontains information to set default values and range restrictions when creating accounts and groups in Security Manager. It contains fields in the following format:

`default_profile_local:<AcctGrp>:<ProfileName>`
The default local profile appears in the default profile field of the new user dialog when creating a local account. The delivered value is set to `System Admin:SA Default`.

`default_profile_global:<AcctGrp>:<ProfileName>`
The default global profile appears in the default profile field of the new user dialog when creating a global account. This value must be set by the system administrator after an appropriate global profile has been defined for the system.

`uid_min_local:<integer>`
This defines the minimum allowed value for the numeric user ID when creating a local account. It should never be set below 100. The delivered value is 1500.

`uid_max_local:<integer>`
This defines the maximum allowed value for the numeric user ID when creating a local account. The delivered value is 2999.

`gid_min_local:<integer>`
This defines the minimum allowed value for the numeric group ID when creating a local group. It should never be set below 100. The delivered value is 1500.

`gid_max_local:<integer>`
This defines the maximum allowed value for the numeric group ID when creating a local group. The delivered value is 2999.

`uid_min_global:<integer>`
This defines the minimum allowed value for the numeric user ID when creating a global account. It should never be set below 100. The delivered value is 3000.

`uid_max_global:<integer>`
This defines the maximum allowed value for the numeric user ID when creating a global account. The delivered value is 9999.

`gid_min_global:<integer>`
This defines the minimum allowed value for the numeric group ID when creating a global group. It should never be set below 100. The delivered value is 3000.

`gid_max_global:<integer>`
This defines the maximum allowed value for the numeric group ID when creating a global group. The delivered value is 9999.

`nis_path:<UNIX_directory_path>`
This specifies the location of the NIS (YP) source files. The delivered value is `/usr/etc/yp/src`.

`modify_accounts:<"true"|"false">`
This determines if Security Manager has the capability to modify UNIX accounts and groups. Security Manager will be able to modify the profile database regardless of the setting. The value should be set to `false` if another account management tool is being used. The delivered value is `true`.

The numeric ranges for the user ID and group ID must be defined but may overlap if necessary. A range of group IDs probably will be reserved in the future for Account Groups and Context-Based Directories. The minimum values for the numeric ID should not be set below 100 to avoid conflict with other essential UNIX system groups and accounts.

C.1 Character and Field Length Limits

Table 3 shows allowed character sets and numeric range limits for account, group, and profile input fields are restricted as follows:

Accounts	Restrictions
Login Name	8-character maximum length. Non-printable characters, backslashes, colons, and spaces are not allowed. Must have at least one character. Must be unique.
Full Name	40-character maximum length. Non-printable characters, backslashes, and colons are not allowed. Must have at least one character.
Password	10-character maximum length. Must have at least one character.
User Number	Must be a positive integer meeting the range restrictions set by the Security Manager configuration file. Must be unique.
Default Profile	The field should be normally completed using the pop-up menu selection. Field length is the combination of the account group name (20 characters), a separating colon, and the profile name length (50 characters). Any entered value must be a valid account group/profile name combination.
Groups	
Group Name	8-character maximum length. Non-printable characters, backslashes, colons, and spaces are not allowed. Must have at least one character. Must be unique.
Group Number	Must be a positive integer meeting the range restrictions set by the Security Manager configuration file. Group Numbers should be less than 60000 for compatibility purposes.
Profiles	
Profile Name	50-character maximum length. Non-printable characters, backslashes, and colons are not allowed. Must be unique for the Account Group to which it belongs.

Table 3. Account, Group, and Profile Input Field Restrictions

Passwords on new accounts are not restricted to the 6-character length and other character restrictions when being created via Security Manager because it is a privileged application. Normal users are subject to minimum password requirements when changing their passwords.

C.2 Scope Restrictions

Accounts and the profiles to which they are assigned must be of the same scope. Local accounts may not be assigned to global profiles, nor may global accounts be assigned to local profiles.

Local and global account management also is affected by the status of NIS on the workstation. In general, the Security Manager always can modify local accounts and profiles on the workstation on which it is executed. If executed remotely on a NIS server and displayed on a client workstation, the Security Manager may only modify the global accounts and profiles administered by the NIS server. It may not modify local accounts and profiles on the NIS server in this case. Table 4 indicates which databases the Security Manager may access, based on the workstation type.

	Local Account	**Local Profile**	**Global Account**	**Global Profile**
Standalone	/	/		
NIS Server	/	/	/	/
NIS+ Server	/	/	/	/
NIS Client	/	/		/
NIS+ Client*	/	/	/	/
Remote**			/	/

Table 4. Databases a Security Manager May Access

* The NIS+ client must be registered as a member of the NIS+ table administration group, otherwise it is treated as a NIS client.

** Security Manager is actually executed on the NIS+ server, with the window being displayed on the invoking workstation's console.

C.3 UNIX System Limitations

On some systems, the number of active groups to which an account has access is limited to 16. An account may be listed with membership in greater than 16 groups but will not be recognized as a member of any group past the sixteenth.

This page intentionally left blank.